Hardware Prediction of OS Run-Length For Fine-Grained Resource Customization

David Nellans, Kshitij Sudan, Rajeev Balasubramonian, Erik Brunvand
School of Computing, University of Utah
Salt Lake City, Utah 84112
{dnellans, kshitij, rajeev, elb}@cs.utah.edu

Abstract—**In the past ten years, computer architecture has seen a paradigm shift from emphasizing single thread performance to energy efficient, throughput oriented, chip multiprocessors. Several studies have suggested that it may be worthwhile to off-load execution of the operating system (OS) to one or more of these cores, or reconfigure hardware during OS execution. To be effective, these techniques must balance the cost of off-loading or re-configuration, versus the potential benefits, which are typically unknown at decision time. These decision points are typically implemented by manually instrumenting a few OS routines (out of hundreds). Such a preliminary research effort cannot be sustained across several operating systems and hardware configurations. We argue that decisions made in software are often sub-optimal because they are expensive in terms of run-time overhead and because applications vary in their use of OS features. We propose that these decision mechanisms should be supported through a hardware based OS run-length predictor, that removes the onus from OS developers. Our final design results in a 95% prediction accuracy for OS intensive applications, while requiring only 2 KB of storage.**

I. Introduction

In the era of plentiful transistor budgets, it is expected that processors will accommodate tens to hundreds of processing cores. Given the abundance of cores, it may be beneficial to allocate some chip area for special-purpose hardware that is customized to execute common code patterns. One such common code is the operating system (OS). Hardware customization for OS execution has high potential because the OS can constitute a dominant portion of many important workloads such as webservers, databases, and middleware systems [2], [5], [6].

Prior studies [2], [4], [5] have advocated that selected OS system calls be off-loaded to a specialized OS core. This can yield performance improvements because (i) user threads need not compete with the OS for cache/ CPU/ branch predictor resources, and (ii) OS invocations from different threads interact constructively at the shared OS core to yield better cache and branch predictor hit rates. Further, in a heterogeneous chip multiprocessor, the OS core could be customized for energy-efficient operation [4], [5] because several modern features (such as deep speculation) have been shown to not benefit OS codes. An alternate technique, with similar goals, reconfigures the hardware of an existing processor to disable these aggressive architectural features if they do not benefit the OS or application [3].

Off-loading implementations have been proposed that range from using the OS' internal process migration methods [4], to layering a lightweight virtual machine under the OS to transparently migrate processes [2]. In re-configuration, decisions about when to modify hardware resources are made when serializing instructions within OS routines are encountered. In all previous studies we are aware of, the decision about which OS sequences are candidates for custom execution, has been made in software, utilizing either static offline profiling or developer intuition. This process is both cumbersome and inaccurate. Firstly, there are many hundreds of system calls and it will be extremely time-consuming to manually select and instrument candidate system calls for each possible OS/ hardware configuration. Secondly, OS utilization varies greatly across applications and decisions based on profiled averages will be highly sub-optimal for many applications.

II. Hardware-based Decision-Making

Instead of a software instrumentation process based on profiled analysis, we propose a hardware-based mechanism that simplifies the task of the OS developer and makes high quality decisions about the benefits of customized OS execution with minimal runtime overhead.

A. Hardware Prediction of OS Run-Length

We believe that operating system run-length is the best indicator of whether customization will be beneficial. This is simply because the overhead of customization is amortized better if the OS routine is longer. The length of the OS invocation is usually a direct function of the processor architected state (which captures input to system calls). We therefore propose a new hardware predictor of OS run-length that XOR hashes the values of various architected registers. After evaluating many register combinations, the following registers were chosen for the SPARC architecture: PSTATE (contains information about privilege state, masked exceptions, FP enable, etc.), g0 and g1 (global registers), and i0 and i1 (input argument registers). The XOR of these registers yields a 64-bit value (that we refer to as *AState*) that encodes pertinent information about the type of OS invocation, input values, and the execution environment. Every time there is a switch to privileged execution mode, the AState value is used to index into a predictor table that keeps track of the invocation length the last time such an AState index was observed, as shown in Figure 1.

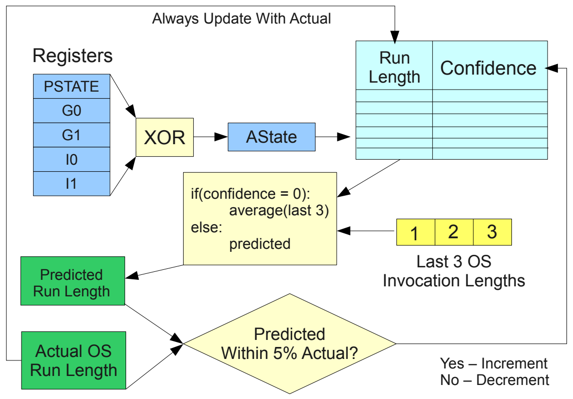

Fig. 1. History based OS run-length predictor functional diagram.

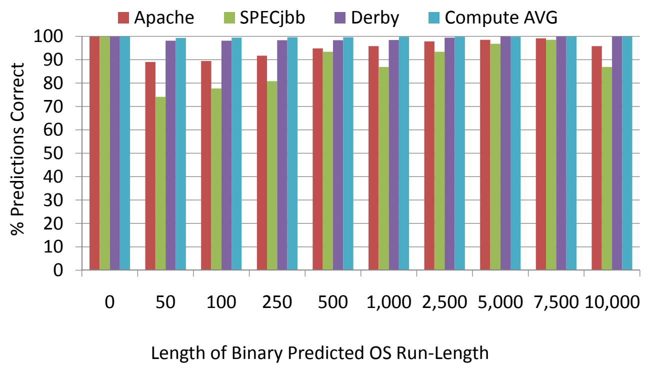

Fig. 2. Accuracy of hardware OS run-length predictions when making binary threshold based decisions.

Each entry in the table also maintains a prediction confidence value, a 2-bit saturating counter that is incremented on a prediction within $\pm 5\%$ of the actual, and decremented otherwise. If the confidence value is 0, we find that it is more reliable to make a "global" prediction, *i.e.*, we simply take the average run length of the last three observed invocations (regardless of their AStates). This works well because we observe that OS invocation lengths tend to be clustered and a global prediction can be better than a low-confidence "local" prediction. For our workloads, we observed that a fully-associative predictor table with 200 entries yields close to optimal (infinite history) performance and requires only 2 KB storage space. The 200-entry table is organized as a CAM with the 64-bit AState value and prediction stored per entry. A direct-mapped RAM structure with 1500 entries also provides similar accuracy and has a storage requirement of 3.3 KB. This table is tag-less and the least significant bits of the AState are used as the index.

Averaged across all benchmarks, this simple predictor is able to precisely predict the run length of 73.6% of all privileged instruction invocations, and predict within $\pm 5\%$ the actual run length an additional 24.8% of the time. Large prediction errors most often occur when the processor is executing in privileged mode, but interrupts have not been disabled. In this case, it is possible for OS execution to be interrupted by one or more additional routines before the original routine is completed. These interrupts typically extend the duration of OS invocations, almost never decreasing it. As a result, our mispredictions tend to underestimate OS run-lengths, resulting in some OS customization possibly not occurring, based on a threshold decision.

B. Binary Decisions and Dynamic Estimation of N

While the hardware predictor provides a discrete prediction of OS run-length, the customization decision must distill this into a binary prediction indicating if the run-length exceeds N instructions and if customization should occur. Figure 2 shows the accuracy of binary predictions for various values of N. For example, if customization should occur only on OS invocation run lengths greater than 500 instructions, then our predictor makes the correct customization decision 94.8%, 93.4%, 96.8%, and 99.6% of the time for Apache, SPECjbb2005, Derby and the average of all compute benchmarks, respectively. While more space-efficient prediction algorithms possibly exist, we observe little room for improvement in terms of predictor accuracy.

The second component of a hardware assisted customization policy is the estimation of N that yields optimal behavior in terms of say, performance or energy-delay product (EDP). For this estimation of N, we rely on algorithms described in past work to select an optimal hardware configuration [1]. If the hardware system must select one of a few possible N thresholds at run-time, it is easiest to sample behavior with each of these configurations at the start of every program phase and employ the optimal configuration until the next program phase change is detected. For our experiments, we use very coarse-grained values of N (0, 100, 1,000, 5,000, 10,000). Increasing the resolution at which N can vary will increase the adaptability of the system, but it comes at the expense of increased sampling overhead.

In example experiments, using L2 cache hit-rate as the feedback metric, our hardware predictor had an average run-time overhead of $<1\%$ when instrumenting all possible OS entry points, compared to 22.8% for a similar software based implementation.

References

[1] R. Balasubramonian, S. Dwarkadas, and D. Albonesi, "Dynamically Managing the Communication-Parallelism Trade-Off in Future Clustered Processors," in *Proceedings of ISCA-30*, June 2003, pp. 275–286.

[2] K. Chakraborty, P. M. Wells, and G. S. Sohi, "Computation Spreading: Employing Hardware Migration to Specialize CMP Cores On-the-Fly," in *ASPLOS-XII*. New York, NY, USA: ACM, 2006, pp. 283–292.

[3] T. Li, L. John, A. Sivasubramaniam, N. Vijaykrishnan, and J. Rubio, "Understanding and Improving Operating System Effects in Control Flow Prediction," *SIGOPS - Operating Systems Review*, vol. 36, no. 5, pp. 68–80, December 2002.

[4] J. C. Mogul, J. Mudigonda, N. Binkert, P. Ranganathan, and V. Talwar, "Using Asymmetric Single-ISA CMPs to Save Energy on Operating Systems," *IEEE Micro*, vol. 28, no. 3, pp. 26–41, May-June 2008.

[5] D. Nellans, R. Balasubramonian, and E. Brunvand, "A Case for Increased Operating System Support in Chip Multi-Processors," in *Proceedings of the 2nd IBM Watson Conference on Interaction between Architecture, Circuits, and Compilers*, September 2005.

[6] J. Redstone, S. J. Eggers, and H. M. Levy, "An Analysis of Operating System Behavior on a Simultaneous Multithreaded Architecture," in *ASPLOS*, 2000.